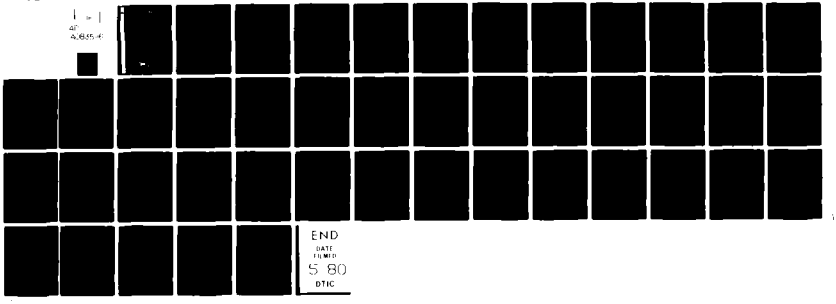
AD-A083 596 CARNEGIE-MELLON UNIV PITTSBURGH PA MANAGEMENT SCIENC--ETC F/G 12/1
A RECURSIVE METHOD FOR SOLVING ASSIGNMENT PROBLEMS,(U)
OCT 79 G L THOMPSON N00014-76-C-0932
UNCLASSIFIED MSRR-448 NL



END
DATE FILMED
5-80
DTIC

AD A083596

LEVEL II

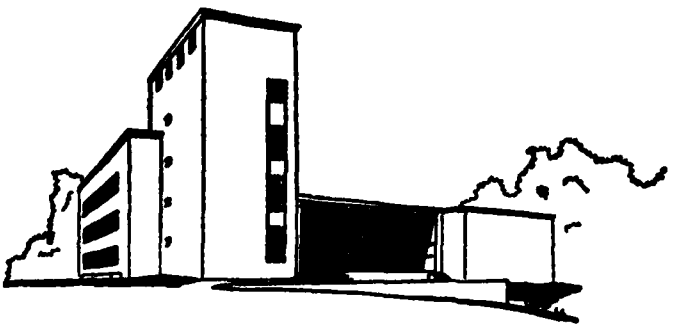
DTIC
ELECTE
APR 2 5 1980
C

Carnegie-Mellon University

PITTSBURGH, PENNSYLVANIA 15213

GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION

WILLIAM LARIMER MELLON, FOUNDER



This document has been approved for public release and sale; its distribution is unlimited.

DDC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MSRR #448
2. GOVT ACCESSION NO.: AD-A083 596
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): A RECURSIVE METHOD FOR SOLVING ASSIGNMENT PROBLEMS
5. TYPE OF REPORT & PERIOD COVERED: Technical Report 10/79
6. PERFORMING ORG. REPORT NUMBER: MSRR #448

7. AUTHOR(s): Gerald L. Thompson
8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0932, N00014-75-C-0621

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR 407-408

11. CONTROLLING OFFICE NAME AND ADDRESS:
Personnel and Training Research Programs
Office of Naval Research (Code 434)
Arlington, Virginia 22217

12. REPORT DATE: 10/79
13. NUMBER OF PAGES: 40

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

11 Oct 79

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

12 45

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

14 MSRR-448

18. SUPPLEMENTARY NOTES

9 Management Science research rept.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Recursive algorithm, Assignment problems, Polynomially bounded algorithm, Linear programming.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The recursive algorithm is a polynomially bounded nonsimplex method for solving assignment problems. It begins by finding the optimum solution for a problem defined from the first row, then finding the optimum for a problem defined from rows one and two, etc., continuing until it solves the problem consisting of all the rows. It is thus a dimension expanding rather than an improvement method such as the simplex. During the method the row duals are non-increasing and the column duals non-decreasing. (continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

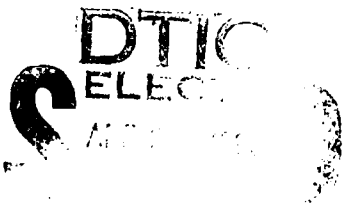
403426

Best and worst case behavior is analyzed. It is shown that some problems can be solved in one pass through the data, while others may require many passes. The number of zero shifts (comparable to degenerate pivots in the primal method) is shown to be at most ~~$n^2/2$~~ n-squared/2.

Extensive computational experience on the DEC-20 computer shows the method to be competitive for at least some kinds of assignment problems.. Further tests on other computers are planned.

(1)

Management Sciences Research Report No. 448



A RECURSIVE METHOD FOR SOLVING ASSIGNMENT PROBLEMS

by

Gerald L. Thompson

October 1979

This report was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under contracts N00014-76-C-0932 and N00014-75-C-0621 NR 047-048 with the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U. S. Government.

Management Sciences Research Group
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

This document has been approved for public release and sale; its distribution is unlimited.

A RECURSIVE METHOD FOR SOLVING ASSIGNMENT PROBLEMS

by

Gerald L. Thompson

ABSTRACT

The recursive algorithm is a polynomially bounded nonsimplex method for solving assignment problems. It begins by finding the optimum solution for a problem defined from the first row, then finding the optimum for a problem defined from rows one and two, etc., continuing until it solves the problem consisting of all the rows. It is thus a dimension expanding rather than an improvement method such as the simplex. During the method the row duals are non-increasing and the column duals non-decreasing.

Best and worst case behavior is analyzed. It is shown that some problems can be solved in one pass through the data, while others may require many passes. The number of zero shifts (comparable to degenerate pivots in the primal method) is shown to be at most $n^2/2$.

Extensive computational experience on the DEC-20 computer shows the method to be competitive for at least some kinds of assignment problems. Further tests on other computers are planned.

Key Words: Recursive algorithm, Assignment problems, Polynomially bounded algorithm, Linear programming.

A RECURSIVE METHOD FOR SOLVING ASSIGNMENT PROBLEMS

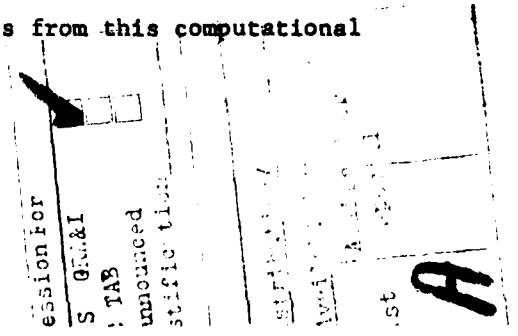
by

Gerald L. Thompson

1. Introduction

The recursive algorithm is a nonsimplex method for solving assignment problems. It begins by finding the optimum solution for a problem defined from the first row, then finding the optimum for a problem defined from rows one and two, etc., continuing until it obtains the optimum for the problem containing all the rows. It is thus a dimension-expanding rather than an improvement method such as the simplex. Throughout the steps of the method the dual variables vary monotonically, with the row duals being non-increasing and column duals non-decreasing.

In this paper it is shown that in the best case the recursive method is a one data pass algorithm for some special kinds of assignment problems. That is, the method can solve such problems after looking each piece of data exactly once. (This should be compared with simplex type methods which require several data passes to get feasible starting solutions, additional passes to choose pivots, and a final complete data pass to verify optimality.) Although the classes of problems for which these one data pass solutions are possible are not very interesting, computational experience is presented to show that more interesting problems can be solved by the recursive method with only slightly more than one data pass. Because only a relatively slow computer was available for testing, the number of data passes as well as computational times were recorded and presented here. It appears from this computational



experience that the recursive method is competitive with other methods on most problems and unbeatable by other methods for the one pass and near one pass problems.

Worst case analysis is also presented which shows that the worst case bound for the recursive method is a polynomial which is less than half of the bound for other polynomially bounded assignment algorithms such as Ford-Fulkerson [5], Balinski-Gomery [1], and Srinivasan-Thompson [10]. Examples which exhibit both best and worst case bounds are presented.

This paper came out of earlier work by the author on auctions and market games [11,12] and is based on previous work in these areas by Shapley and Shubik [7] and Barr and Shaftel [3]. The idea of solving bottleneck assignment problems by adding rows one by one has been previously used by Derigs and Zimmermann [4]. The backshift operation used in this paper is similar to analogous operations in the Derigs -Zimmermann paper and also to the one in the relaxation method of Hung and Rom [6]. Finally, the alternating path basis, which is shown in this paper to correspond to Ordens perturbation method, and also to the dutch auction procedure of Barr and Shaftel, was previously employed by Barr, Glover, and Klingman [2].

The recursive method has been extended to transportation problems, and to bottleneck assignment and transportation problems. Those extensions will be presented elsewhere.

2. Notation and Preliminary Results

We consider the assignment problem to be a market game with the rows representing sellers and the columns buyers. The index sets of the sellers and buyers are

$$I' = \{1,\ldots,n\} \tag{1}$$

$$J' = \{1,\ldots,n\} \tag{2}$$

We assume each seller has one unit to sell and each buyer wants to buy one unit. Let

$$c_{ij} \geq 0 \tag{3}$$

be the bid of buyer j for seller i's good. For technical reasons we add a dummy seller and a dummy buyer so that the index sets (1) and (2) become

$$I = I' \cup \{n+1\} = \{1,\ldots,n,n+1\} \tag{4}$$

$$J = J' \cup \{n+1\} = \{1,\ldots,n,n+1\} \quad . \tag{5}$$

The bids for these dummy players are

$$c_{n+1,j} = 0 \quad \text{for} \quad j\epsilon J, \quad \text{and} \quad c_{i,n+1} = 0 \text{ for } i\epsilon I. \tag{6}$$

The assignment problem can now be stated as

$$\text{Maximize} \quad Z = \sum_{i\epsilon I} \sum_{j\epsilon J} c_{ij}x_{ij} \tag{7}$$

$$\text{Subject to} \quad \sum_{j\epsilon J} x_{ij} = 1 \quad \text{for} \quad i\epsilon I' \tag{8}$$

$$\sum_{i\epsilon I} x_{ij} = 1 \quad \text{for} \quad j\epsilon J' \tag{9}$$

$$\sum_{j\epsilon J} x_{n+1,j} = 0 \tag{10}$$

$$\sum_{i\epsilon I} x_{i,n+1} = 0 \tag{11}$$

$$x_{ij} \geq 0 \quad . \tag{12}$$

It is well known that the assignment problem is massively (primal) degenerate. One way of handling this problem is to perturb the right hand sides of (8) and (11) as follows:

$$(P1)\quad \begin{cases} \sum_{j\in J} x_{ij} = 1 + \epsilon \quad \text{for} \quad i\in I' & (8') \\ \sum_{j\in J} x_{n+1,j} = \epsilon & (10') \\ \sum_{i\in I} x_{i,n+1} = (n+1)\epsilon & (11') \end{cases}$$

where $0 \leq \epsilon < 1/2(n+1)$. We call this perturbation (P1). (Another perturbation (P2) which adds ϵ to the right hand sides of (9) and (11) and $(n+1)\epsilon$ to the right hand side of (10) leads to an equivalent algorithm which we do not discuss here.)

The following facts are well known. For a small positive ϵ, a <u>basic</u> feasible solution to (8'), (9), (10') and (11') has exactly $2n+1$ cells (i,j) with $x_{ij} > 0$; all other $x_{ij} = 0$. If B is the set of basis cells then the graph $G = (I \cup J, B)$ is a <u>tree</u>. Let $X(\epsilon)$ be a basic solution, let x_{ij} be the optimal shipping amount for basis cell (i,j), at let $R(x_{ij})$ be the scientifically rounded value of x_{ij}; if $R(x_{ij}) = 1$ then we say cell (i,j) ships 1, and if $R(x_{ij}) = 0$ then we say cell (i,j) ships 0.

<u>Lemma 1</u>. Given perturbation (P1) a feasible solution is basic if and only if each row (except n+1) has exactly 2 basis cells, one shipping 1 and the other shipping 0.

<u>Proof</u>. Each row $i\in I'$ has a supply of $1+\epsilon$ and row $n+1$ has a supply of ϵ, while each column $j\in J'$ has a demand of 1 and column $n+1$ has demand of $(n+1)\epsilon$. Hence each row $i\in I'$ must have at least two basis cells to use up its supply, and row n+1 must have at least one basis cell. Since this adds up to $2n+1$ basis cells, each row $i\in I'$ has exactly 2 basis cells and row $n+1$ has 1. To show the rest of the statement, recall that every tree has at least one pendant node, i.e., there is at least one cell (i,j) which

is the only basis cell in column j for some j. Since the solution is assumed feasible, $x_{ij} = 1$ i.e., (i,j) ships 1. The other basis cell in row must therefore ship 0. Now cross out row i and column j and repeat the same argument for the rest of the rows.

We now introduce some graph terminology. Assume the tree $G = (I \cup J, B)$ is drawn with row $n+1$ as the root, see [8], which appears as the node at the top of the tree, and with downward pointing arcs going to other nodes. Let a downward arc be the son (or successor) relation and the reverse relation be the father (or predecessor) relation. All nodes, except row $n+1$, have fathers.

Definition. A basis tree has the unique row-son property if every row node (except possibly the root node) has one and only one son.

Lemma 2. Given perturbation (P1).

(a) The graph of a feasible solution has the unique row son property.

(b) The graph of a feasible solution is an alternating path basis in the sense of Barr, Glover, and Klingman [2].

Part (a) can be proved by using the crossing out routine given in the proof of Lemma 1. The unique row-son property is simply another way of stating definition of an alternating basis, see [2].

The dual problem to the perturbed problem (7), (8′), (9), (10′), (11′), (12) is

$$\text{Minimize} \quad \{ \sum_{i \in I'} u_i(1+\epsilon) + u_{n+1}\epsilon + \sum_{j \in J'} v_j + v_{n+1}(n+1)\epsilon \} \qquad (13)$$

Subject to

$$u_i + v_j \geq c_{ij} \quad \text{for} \quad i \in I, \quad j \in J \, . \qquad (14)$$

Because of assumption (3), it can be shown (see [11]) that we can also impose nonnegativity constraints

$$u_{n+1} \geq 0 \quad \text{and} \quad v_{n+1} \geq 0 \quad . \tag{15}$$

This, together with (6), implies $u_{n+1} = v_{n+1} = 0$ so that (13) becomes

$$\text{Minimize } \{[\sum_{i\epsilon I'} u_i + \sum_{j\epsilon J'} v_j] + \epsilon \sum_{i\epsilon I'} u_i\} \ . \tag{16}$$

Definition. We call u_i the price for seller i, and v_j the surplus of buyer j.

Lemma 3. The solution of the problem with perturbation (P1) simultaneously

(a) Minimizes the sum of the sellers prices;

(b) Maximizes the sum of the buyers surpluses.

Proof. An optimal solution to (7), (8′), (9), (10′), (11′) and (12) for $\epsilon > 0$ remains optimal when ϵ is replaced by 0, see [10]. Let Z be the value of the optimal solution when $\epsilon = 0$. By the duality theorem if u_i and v_j are optimal solutions to (14), (15), and (16) when $\epsilon = 0$

$$Z = \sum_{i\epsilon I'} u_i + \sum_{j\epsilon J'} v_j \ . \tag{17}$$

Substituting (17) in to (16) gives for $\epsilon > 0$

$$\text{Minimize } \{Z + \epsilon \sum_{i\epsilon I'} u_i\} \tag{18}$$

which, since Z is constant, proves (a). Solving (17) for Σu_i and substituting into (18) gives

$$\text{Minimize } \{Z(1+\epsilon) - \epsilon \sum_{j\epsilon J'} v_j\} \tag{19}$$

which proves (b).

The set of all solutions to (14), (15), and (16) with $\epsilon = 0$ is called the core of the game by Shapley and Shubik [7]. In [11], the author gives an algorithm for finding all basic solutions in the core, starting with the maximum buyer surplus solution of Lemma 3.

One last bit of terminology is appropriate. A dutch auction is a concealed bid auction in which the price (indicated by a "clock") starts high and steadily drops; when the highest bidder makes his (concealed) bid he is certain to get the object being sold; however, the price continues to drop until the second highest bidder makes his (concealed) bid which "stops the clock"; the price the highest bidder pays is therefore determined by the bid of the second highest bidder. The solution procedure to be proposed is formally like a dutch auction. As we saw above, the perturbation (P1), the solution has two basis cells in each row one shipping 1, corresponding to the highest bidder who gets the good, and the other shipping 0, corresponding to the second highest bidder who determines the price. In the description of the algorithm of the next section, economic as well as mathematical terminology will frequently be used because it provides intuitive reasons as to why the procedures are carried out as they are.

3. Outline of the Recursive Method

As noted earlier, the recursive method is not a simplex method even though it uses some of the terminology and concepts of that method. In order to emphasize this difference we state two characteristics of the recursive method which are distinctly different from those of the simplex method.

(a) In the recursive method every feasible primal or dual solution found is also optimal.

(b) In the recursive method a primal solution is not proved to be optimal by means of the dual; in fact, it is possible to find an optimal primal solution to an assignment problem without ever finding its optimal dual solution.

Definition. Problem P_k consists of the first k sellers, the dummy seller $n+1$ who is given $n-k$ dummy units to sell, and all the buyers including the dummy buyer. In other words, problem P_k can be obtained from (7) - (12) by replacing I by $I_k = \{1,\ldots k,n+1\}$, and the right hand side of (10) by $n-k$.

A brief outline of the recursive method is: solve P_0; from its solution find the solution to P_1; from P_1 solve P_2; etc.; continue until the solution to P_n is produced from the solution to P_{n-1}.

A more detailed outline is given in Figure 1. As indicated we first find the optimal primal and dual solutions to P_0 which is the trivial one row problem consisting of the slack row only. From them we find the optimal primal solution to P_1, which consists of the first row of the assignment problem, and the slack row. Then we find the optimal dual solution to P_1. Then we find the optimal primal solution to P_2 followed by its optimal dual solution. Etc. Finally, the optimal primal solution to P_n is constructed from the optimal solution to P_{n-1}. As noted in Figure 1, the actual finding of the optimal dual solution to P_n is optional, and need not be carried out if only the optimal primal solution is wanted. As noted in the figure there are two recursive steps A and B, where step A constructs the optimal primal solution to P_k from the optimal solution to P_{k-1}, and step B constructs the optimal dual solution to P_k from its optimal primal. These two steps will be discussed in detail in the next two sections. In the remainder of this section we discuss the solutions of P_0 and P_1.

The initial basis graph for P_0 is shown in Figure 2. It corresponds to the solution

$$x_{n+1,j} = 1 \text{ for } j \in J', \quad x_{n+1,n+1} = 0, \quad u_{n+1} = 0 \text{ and } v_j = 0 \text{ for } j \in J.$$

In order to solve P_1 we find the two largest reduced bids in row 1. Let C_ℓ and C_s be columns in which the largest and second largest reduced bids are, i.e., since all $v_j = 0$,

$$c_{1\ell} = \underset{j \in J}{\text{Max}}\, c_{1j}, \quad c_{1s} = \underset{j \in J-\{\ell\}}{\text{Max}}\, c_{1j}$$

We then perform an easy back shift to get the optimal primal and dual solutions for P_1 as illustrated in Figure 3. First node C_ℓ is cut out of the P_0 basis graph; then node R_1 is made the son of C_s, node C_ℓ the son of R_1, and finally we set $u_1 = c_{1s}$ and $v_\ell = c_{1\ell} - c_{1s}$. As is evident, only a few instructions are needed to perform the easy back shift. Also it is easy to check that the indicated solution is optimal for problem P_1.

Surprisingly, it turns out that the same easy back shift operation suffices to solve many of the sub problems. As will be seen in Section 8 on computational experience, for most problems from 40 to 95 percent or more of the sub problems P_k and be solved in this extremely simple fashion.

4. Recursive Step A

The purpose of recursive step A is: from the known basic optimal primal and dual solutions to problem P_{k-1} find an optimal primal solution to P_k. This will be done by carrying out a back shift step, to be defined next. Recall that the basis tree T_{k-1} for P_{k-1} is always stored with node R_{n+1} as the root.

The (Ordinary) Backshift Step.

1. In one pass through row k find the two largest reduced cost entries; i.e.

$$c_{k\ell} - v_\ell = \underset{j \in J}{\text{Maximum}} \ (c_{kj} - v_j)$$

$$c_{ks} - v_s = \underset{j \in J-\{\ell\}}{\text{Maximum}} \ (c_{kj} - v_j) \ .$$

2. Set $u_k = c_{ks} - v_s$.

3. Add node k and arc (k,ℓ) to T_{k-1}.

4. Suppose the backward path (found by applying the father relation) from R_k to R_{n+1} is $R_k, C_\ell, \ldots, C_p, R_{n+1}$. Interchange fathers and sons along this backward path from R_k to C_p.

5. Cut (remove) arc $(p,n+1)$ from T_{k-1}.

A picture of the ordinary backshift step is shown in Figure 4. Note that the unique row son property is preserved in the relabelling of Step 4. Notice also that at Step 5 of the backshift step the tree is broken into two subtrees: tree T_k^* which has root R_k, and tree T_{n+1} which has root R_{n+1}. Also, as marked in the figure, because no dual variables are changed during the backshift step, all arcs are dual feasible except (possibly) for the new arc (k,ℓ).

The comparison of the backshift operation with a primal pivot step is instructive. The search for the incoming cell in Step 1 is confined to a single row; there is no cycle finding; there is no search for the outgoing cell, since it is known in advance to be $(p,n+1)$; finally no dual changes are made during the backshift operation. Hence the backshift step requires less than half the work of a primal pivot.

Counting easy backshifts, there are always exactly n backshift steps made in the course of solving an $n \times n$ assignment problem. As will be seen empirically in Section 8, from 40-95 percent of the backshifts are easy and the rest are ordinary. A final remark is that the backshift step is the only operation of the recursive method in which the father-son relationships at a node are interchanged.

Lemma 4. The backshift operation produces a primal feasible solution whose objective function satisfies

$$Z_k = Z_{k-1} + (c_{k\ell} - v_\ell) \tag{20}$$

where Z_h is the objective function value for problem P_h and v_ℓ is the dual variable for column ℓ in problem P_{k-1}.

Proof. The fact that the backshift step preserves the unique row-son property and hence produces a primal feasible solution was noted above. Let s and s' be the son functions for the solutions of P_{k-1} and P_k, respectively, let f be the father function for P_{k-1}, let $I_h = \{1,\ldots,h\}$, and let B_{k-1} be the set of rows on the backward path in T_{k-1} from node C_ℓ to node C_p. Then we have

$$\begin{aligned} Z_k - Z_{k-1} &= \sum_{h \in I_k} c_{h,s'(h)} - \sum_{h \in I_{k-1}} c_{h,s(h)} \\ &= c_{k,\ell} + \sum_{h \in I_{k-1}} [c_{h,s'(h)} - c_{h,s(h)}] \\ &= c_{k,\ell} - \sum_{h \in B_{k-1}} [c_{h,s(h)} - c_{h,f(h)}] \\ &= c_{k,\ell} - v_\ell \, . \end{aligned}$$

The third step follows from the fact that $s'(h) = s(h)$ for $h \notin B_{k-1}$ and $s'(h) = f(h)$ for $h \in B_{k-1}$. The last step follows from the facts that $u_{n+1} = v_p = 0$ in the solution to P_{k-1}, and v_ℓ can be computed by working down the tree from node C_p to node C_ℓ.

Theorem 1. The back shift step produces a primal optimal solution to Problem P_k.

Proof. We present two proofs of this theorem. The first rests on an economic argument which some readers may not find completely convincing. The second proof, however, rests on standard linear programming reasoning.

Economic Optimality Proof. Think of the change from P_{k-1} to P_k as that of an auction in which one more seller is added. The buyers positions cannot be worsened by this change, i.e., buyer surpluses v_j will stay the same or increase. Consequently buyer j's bid in P_k for seller k's good is $c_{kj} - v_j$, i.e., it is his original bid c_{kj} less his current surplus v_j. The maximum change $\Delta Z_{k-1} = Z_k - Z_{k-1}$ that can be made in the objective function is obtained by "selling to the highest bidder," buyer ℓ. By Lemma 4, this sale actually produces the maximum possible change $c_{k\ell} - v_\ell$ in the objective function, hence the theorem.

Duality Proof. As will be shown in the next section (see Theorem 2), recursive step B produces in a finite number (at most k) of steps a feasible dual solution to P_k corresponding to the primal solution obtained from the backshift step. From the duality theorem of linear programming both the primal and dual solutions to P_k are therefore optimal.

5. Recursive Step B.

The purpose of recursive step B is: from the primal solution to P_k

found at the end of recursive step A, construct a basic (maximal buyer surplus) dual feasible (hence optimal) solution to P_k without changing the primal solution. As can be seen in Figure 4, this step has two objectives: (a) to make arc (k,ℓ) dual feasible, and (b) to join together the trees T_k^* and T_{n+1}.

To help intuition concerning recursive step B it is best to revert to economic terminology. In Figure 4, the row and column nodes in tree T_{n+1} represent sellers and buyers whose prices and surpluses, respectively, are correct for problem P_k; however, the sellers (row nodes) in tree T_k^* have prices which are (possibly) too high while the buyers (column nodes) in T_k^* have surpluses which are (possibly) too low.

In the first part of recursive step B, called step B1, we simultaneously reduce all prices for sellers in T_k^* and increase all surpluses for buyers in T_k^*. As prices drop, a buyer in T_{n+1} may become a second highest bidder for the good of a seller in T_k^*. Then, as illustrated in Figure 5, part of tree T_k^* is removed and attached to T_{n+1}. This is called a <u>zero shift</u> operation. Recursive step B1 continues with zero shifts being made as needed until it is seller k whose good has a second highest bidder in tree T_{n+1}. At this point, no zero shift is made, and recursive step B2 is initiated. Note that the infeasibility of arc (k,ℓ) nas not changed during step B1. What has been accomplished during this step is that the correct selling price u_k for seller k has been determined.

During recursive step B2 the selling price u_k is held fixed while all other sellers' prices in T_k^* are reduced, and all buyer surpluses v_j for j in T_k^* are increased simultaneously, with zero shifts being made as needed. Since the reduced cost of arc (k,ℓ) is $c_{k\ell} - u_k - v_j$ we see that arc (k,ℓ)

becomes steadily less infeasible as v_j increases. When v_j has become large enough so that arc (k,ℓ) is just feasible, changes in seller prices and buyer surpluses stop, tree T_k^* is pasted to tree T_{n+1} and recursive step B2 ends with dual and primal feasible (hence optimal) solutions to problem P_k.

In order to describe these steps precisely we define two subroutines which depend on a subset of rows, called ROWS, of T_k^* and a subset of columns, called COLS, of T_{n+1}. The first subroutine, SEARCH, is defined as follows:

<u>SEARCH</u> Calculate

$$\lambda = \underset{\substack{i \in \text{ROWS} \\ j \in \text{COLS}}}{\text{Minimum}} \; (c_{ij} - u_i - v_j)$$

Let (r,s) be a cell on which the minimum is taken on.

Let h be the father of r in T_k^*.

Let T_r be the subtree of T_k^* which is below r.

The second subroutine, DUAL, is defined as:

<u>DUAL</u>. Let

$$u_i \rightarrow u_i - \lambda \quad \text{for} \quad i \in \text{ROWS}$$

$$v_j \rightarrow v_j + \lambda \quad \text{for} \quad j \in \text{COLS}.$$

Note that application of the dual operation, which is the only step in which dual variables are changed can only decrease seller prices and increase buyer surpluses.

Given these subroutines we can now define Recursive Step B1.

Step B1

(1) Let ROWS be the set of rows in T_k^*;
let COLS be the set of columns in T_{n+1}.

(2) Use SEARCH to find λ, r, h, T_r.

(3) Change dual variables by applying DUAL.

(4) If r = k go to recursive step B2.
Otherwise go to 5.

(5) Cut arc (r,h) and paste arc (r,s).

(6) Go to (1).

We can define Recursive Step B2 similarly.

Step B2

(1) Let ROWS be the set of rows in T_k^* except for row k; let COLS be the set of columns in T_{n+1}.

(2) Let $\lambda^* = -c_{k\ell} + u(k) + v(\ell)$, where ℓ was found as the largest reduced bid in row k.

(3) Use SEARCH to find λ, r, h, T_r.

(4) If $\lambda \geq \lambda^*$ go to (7); otherwise go to (5).

(5) Change dual variables by applying DUAL;
let $\lambda^* \rightarrow \lambda^* - \lambda$; let $v(\ell) = v(\ell) + \lambda$.

(6) Cut (r,h); paste (r,s). Go to (1).

(7) Let $\lambda \rightarrow \lambda^*$; apply DUAL, let $v(\ell) = v(\ell) + \lambda$.

(8) Paste (k,s).

(9) END

Each time the cut and paste operation is performed in either step B1 or B2 we will say a zero shift has been made. Sometimes during the course of application of the SEARCH subroutine it finds $\lambda = 0$; when this happens it is possible to do the cut and paste operation without making a dual change and continue the SEARCH routine without finding ROWS and COLS again; such a zero shift is called an easy zero shift. Use of easy zero shifts has greatly speeded up the performance of the code, see Section 8.

The maximum number of elements in the set ROWS for either step B1 or B2 is k. Each time the cut and paste step is made at least one row (and at least one column) is transferred from the tree T_k^* to tree T_{n+1} and ROWS is correspondingly made smaller. Therefore k is the maximum number of zero shifts that can be made by Recursive Step B at step k.

Note that when T_k^* consists of just row k and its son, the set ROWS in (1) of Step B2 is empty, so that λ found in (3) is $+\infty$. In this case the algorithm is certain to terminate since $\lambda = \infty > \lambda^*$ in (4). Also when (8) of Step B2 is made, a basic feasible dual solution has been found.

We summarize the above in a theorem.

Theorem 2. Given the primal feasible solution to P_k produced by Recursive Step A, the application of Recursive Steps B1 and B2 will produce a basic feasible dual solution to P_k after making at most k zero shifts. During the application of recursive step B the seller prices are non increasing and the buyer surpluses are non decreasing.

6. Solution of an Example

Consider the 3X3 example shown in Figure 6(a) in which a slack row and column have also been added. Figure 6(b) shows the solution to problem P_0,

where the basis cells are circled, and the shipping amounts for each basis cell are marked above the circle. Dual variables are marked to the left and the top of the figure. These conventions hold for the rest of the figures as well.

Since 17 and 15 are the largest and second largest bids in the first row of the problem, the easy backshift operation applied to P_1 yields the optimal solution in Figure 6(c); the optimal basis tree for P_1 is shown in Figure 7(a).

As can be seen in Figure 6(d) the two highest bids in row 2 of P_2 are 22 and 19, because the reduced cost in column 3 is 18. The situation at the end of the first backpivot is shown in Figure 6(d). The corresponding two parts of the basis tree are shown in Figure 7(b). Because T_k^* has only row k in it recursive step B1 is empty, and step B2 can be carried out by pasting R2 to C1 by means of arc (2,1); also the λ in figure 6(d) is set equal to 3 as indicated in (5) of B2. The resulting tableau is shown in Figure 6(e) and the optimal basis tree appears in Figure 7(c).

When the third row of the original problem is added, it can be seen that the highest reduced bid of 18 occurs in column 2 and the second highest bid of 17 is in column 1. The backpivot step involves adding R3 as a son of C2 in Figure 7(c) relabelling the tree back to C1, and cutting arc (4,1). The resulting two parts of the basis tree appear in Figure 7(d). The tableau at the end of the backpivot step appears in Figure 6(f). Note that the shipping amounts have changed as follows: x_{22} from 1 to 0, x_{21} from 0 to 1, and x_{41} from 1 to 0. The actual transfers of goods from sellers to buyers are indicated by the basis cells which ship 1 in Figure 6(f). These transfers are, in fact, the optimum assignment as proved in Theorem 1. Of course, the

dual solution in Figure 6(f) is neither feasible nor basic and we go to recursive steps B1 and B2 to achieve that. The direction of price and buyer surplus changes for step B1 are indicated by the λ's marked in Figure 6(f). Here ROWS = {1,2,3} and COLS = {4}; hence $\lambda = 15$, $r = 1$, $h = 1$, and $s = 4$. Hence we change duals by 15, cut arc (1,1), paste arc (1,4) and arrive after one zero shift at the situation indicated in Figures 6(g) and 7(e). We repeat step B1 with ROWS = {2,3}, COLS = {3,4} and λ changes as marked in Figure 6(g). Now $\lambda = 1$, $r = 2$, $h = 2$, and $s = 3$. Hence we change duals by 1, cut arc (2,2), paste arc (2,3) and arrive at the situation depicted in Figures 6(h) and 7(f). Now ROWS = {3}, COLS = {1,3,4}, $\lambda = 0$, $r = 3$, h is not defined, and $s = 1$. We are now in step B2 with $\lambda^* = 1$ and ROWS and COLS as before. Now $\lambda = \infty$ so we paste arc (3,1) increase v(2) by 1 and come to the optimal tableau in Figure 6(i) and its corresponding optimal basis tree in Figure 7(g).

7. Best Case and Worst Case Analysis

In contrast to many other mathematical programming algorithms, it is easy to determine the best and worst case behavior for the recursive algorithm, and it is also easy to find examples which have either kinds of behavior. In fact, a single example will be given which exhibits both kinds of behavior, depending on the order of introduction of the sellers (rows)! A second example which shows not quite such extreme behavior is also presented.

The numbers of easy backshifts, ordinary backshifts, and zero shifts for the best and worst cases are shown in Figure 8. These numbers for the best case are easy to determine. The maximum number of easy backshifts is n-1, since in the most favorable case all the second highest bidders can be chosen to be in

the same column, and that column is therefore not available for an easy backshift. The minimum number of zero shifts is 1 (at least if the optimum dual to P_n is desired) since at step n the tree T_{n+1} has just two elements R_{n+1} and C_{n+1}, and at least one zero shift is needed to hook T_n^* and T_{n+1} together.

For the worst case analysis, recall that the solution to P_1 involves an easy backshift, and that there are always exactly n backshifts. Hence the worst case involves 1 easy backshift and $n-1$ ordinary backshift. To determine the maximum number of zero shifts recall that no zero shift is needed for P_1; for problem P_k with $k \geq 2$ the maximum number of rows in T_k^* after the backshift is k, and at each zero shift at least one row is shifted from T_k^* to T_{n+1}; therefore the maximum total number of zero shifts is

$$2 + 3 + \ldots + (n-1) + n = \frac{(n+2)(n-1)}{2} \tag{21}$$

as indicated in Figure 8.

To exhibit the various best and worst cases consider the following four examples with $m = n = 100$,

$$c_{ij} = \begin{cases} 1 \ , & \text{if } j \neq 1 \\ 101-i, & \text{if } j = 1 \end{cases} \tag{22}$$

$$c_{ij} = \begin{cases} 1 \ , & \text{if } j \neq 1 \\ i \ , & \text{if } j = 1 \end{cases} \tag{23}$$

$$c_{ij} = (101-i)(101-j) \tag{24}$$

$$c_{ij} = i \cdot j \ . \tag{25}$$

Note that problems (22) and (23) are related in that the order of listing the rows is reversed. Problems (24) and (25) are similarly related.

The computer solution statistics for these four problems are given in Figure 9. Note that problem (22) is a one-pass problem, while problem (23), which is the same except for order of rows, has the maximum number (5049) of zero shifts which is given by (21) when $n = 100$. However, the area search factor (that is, the ratio of the number of data calls to the number of data elements) which indicates the number of times the data is searched to solve (23), is only 50 percent more than one data pass and more than 98 percent of the zero shifts are easy zero shifts; also the time to solve (23) is more than 10 times that of (22).

Problem (25) is known to be of maximum difficulty for the Hungarian algorithm, and is also of maximum difficulty for the recursive method; as indicated in Figure 9, its solution requires 5049 zero shifts none of which is easy. The time to solve (25) is nearly 640 times as long as the time to solve (22), which is somewhat less than the ratio of the areas searched. Problem (24), which is the same as (25) except for order of rows, is much easier in that it requires 99 zero shifts none of which is easy. The time to solve (25) is 20 times as long as (24), which is somewhat less than the ratio of the areas searched. Note that the costs in problems (24) and (25) range from 1 to 10,000.

Another way to evaluate the recursive method is to compare its worst case behavior with worst case behaviors for three other methods due to Ford-Fulkerson [5], Balinski-Gomery [1], and Srinivasan-Thompson [10], which are known to be polynomially bounded, see Figure 10. The amount of work needed for primal or dual non-breakthrough steps is approximately the same as a primal pivot step in the cost operator method (see [10]) and is somewhat more than a

zero shift step in the recursive method. However the recursive method's worst case behavior is less than half that of other methods.

Comparisons of worst case behavior are not usually good indicators of average case behavior. In the next section some computational results on randomly generated problems is discussed.

8. Computational Experience

The recursive method has been programmed in FORTRAN and extensively tested. The computer used was a DEC-20 which is (a) relatively slow, (b) has a limited memory, and (c) is always operated in a multi-programming environment so that timing results can vary by as much as 10-50 percent on different runs of the same problem.

Table 1 presents computational experience for 150 randomly generated problems having $n = 100, 200,$ or 300 so that the number of nodes $(2n)$ is 200, 400, or 600, and approximately 4000 arcs with arc costs unformly distributed between 0 and the indicated maximum cost. Each row of the table presents average results for 5 randomly generated problems. The maximum cost ranged from 1 to 1000. By observing one of the problem sizes for various maximum costs the so-called "minimum cost effect" (which we will call the "maximum cost effect" since we have a maximizing rather than a minimizing problem) first observed by Srinivasan and Thompson in [9] will be seen; that is, for a fixed number of nodes and arcs, as the maximum cost increases from 1 to 1000 the time and area search factor increase while the percentages of easy backshifts and zero shifts decrease. In other words, for a fixed problem size, the larger the cost range, the harder the problem.

As indicated in the previous section the area search factor is the ratio of the number of times arc cost data is called by the code to the total number of arcs. For instance for $n = 100$ the area search factor in the table is 1.08 (meaning 8 percent of the data is called twice, the rest once) when the maximum cost is 1 and 21.97 when it is 1000. The corresponding computation times are .14 and 1.91. The ratio of the area factors (20.3) is greater than the ratio of the times (13.6) for this example. The same result is true for other cases in the table, but it is clear that the two ratios are highly correlated. Since the area factors are independent of the computer being used, they give a measure of problem difficulty which is independent of the computer being used.

In Table 2 computational experience with 90 problems each having approximately 18,000 arcs and nodes ranging from 400 to 2000 ($n = 200$ to 1000). For each problem with a given set of nodes and arcs, two sets of costs are generated, one with maximum cost 1 and the other with maximum cost 100. The results are shown on successive lines, and indicate that the second type of problem is about 10 times as hard as the first.

Another result that was evident when step by step solution data was printed, was that the easy backshifts tend to occur early in the solution process, i.e., the time needed to solve problems $P_1,\ldots,P_{n/2}$ was much less than the time needed to solve problems $P_{n/2},\ldots,P_n$. In fact the last step of finding the optimal dual solution to P_n always involves at least one and usually many zero shifts. An example of this is shown in Table 3, where cumulative data for the solutions of P_{92} - P_{100} are shown. Note that Problems P_1 - P_{92} are solved by easy backshifts, without entering the main part of the program; P_{93} and P_{94} require 1 ordinary backshift and 1 zero shift each; P_{95} is solved by

an easy back shift; P_{96} requires 1 ordinary backshift and 1 zero shift; P_{97} and P_{98} are solved by easy backshifts; P_{99} requires an ordinary backshift and 20 zero shifts, 19 of which are easy; finally, P_{100} requires 1 ordinary backshift and 2 zero shifts. The total area searched to solve this problem was only 4 percent more than a 1 pass solution would require even though the problem is only 50 percent dense. The ease of its solution is due to the fact that the maximum cost was 1.

Problems which are larger, have larger maximum costs, or which are less dense than the one shown in Table 3, have more complicated solutions than the one illustrated there. Nevertheless their solution behavior is similar with the first few subproblems being very easy to solve and later ones becoming progressively harder.

The author is currently trying to test the recursive algorithm code on faster computers so that meaningful timing comparisons with other algorithms can be made.

9. Practical Coding Considerations

The computational results of the previous section indicate that the recursive method is a competitive algorithm for solving assignment problems. Several considerations should be taken into account in the design of a practical code for this purpose.

First, objective function (7) is maximizing rather than minimizing which is more common. If we change a minimizing objective to a maximizing one by multiplying each cost by -1 then the nonnegativity assumption (3) is violated. It is easy to get around that difficulty by adding 1 minus the most negative cost

to each cost, and suitably adjusting the optimal objective function when found. Of course, finding the most negative cost requires one data pass, unless it is supplied by the user.

Another good idea would be to find the largest entry in each row and introduce the rows in decreasing order of these largest entries. This again would involve one pass through the data (which could be the same as the data pass to find the most negative cost). The evidence from examples (22) - (25) indicate that the value of having a good order in which to introduce rows can be considerable.

The memory requirements for the recursive method can be stated in terms of N the number of nodes and A the number of arcs. For the version of the code which solves completely dense problems (i.e., $A = n^2$) the memory requirements are $4N + A$; and for the sparse version of the code, the memory requirements are $4N + 2A$. These compare favorably with similar requirements for the alternating path algorithm [2]. The number of FORTRAN instructions required for the recursive method is less than half of those needed for the primal code in [9].

Because of the relatively small memory requirements and relatively small running times it is likely that the recursive method will be a good choice for installation in minicomputers.

10. Comparisons with the Simplex Method

Although the recursive method is not a simplex method, it is instructive to make comparisons between the steps of the two methods. A detailed comparison between a primal pivot step and the ordinary backshift and zero shift operations is presented in Figure 11. Note that aside from the search step which is extremely difficult to compare since it is problem dependent as well as algorithm

dependent, both the backshift and zero shift operations involve less work than the primal pivot (roughly one-half as much). This comparison is even more pronounced in the case of easy backshifts and easy zero shifts.

In Table 1(B) of [9] it was reported that the solution by a primal algorithm of a 100 x 100 assignment problem having about 4000 arcs and maximum cost of 100 required 651 primal pivots and 2.187 UNIVAC-1108 seconds. In Table 1 it can be noted that the solution of a similar problem by the recursive method requires 66 ordinary and 44 easy backshifts,and 255.4 zero shifts of which more than 57 percent were easy, and took 1.12 DEC-20 seconds. Comparison of these times on two different machines is very difficult, but it is a common belief that the UNIVAC-1108 is considerably faster than the DEC-20.

The author is undertaking more extensive testing on a single computer of these and other methods. Others have also volunteered to help in this testing.

It should also be noted that the recursive method requires neither a starting solution nor a phase I part of the code. Also it does not require artificial arcs; even for sparse problems. It does not need degeneracy prevention techniques since as noted in Section 2, the dutch auction solution is, in fact, equivalent to the usual well-known perturbation technique.

Another feature of the recursive method is that it is able to take advantage of alternate optimal solutions when selecting the two highest bids in a row. When several choices are possible, it is a good idea to select, when possible, these two highest bids so than an easy backshift can be performed. Note in Table 1 that the number of easy backshifts goes down from over 90 percent to the low 40's as the maximum cost increases from 1 to 1000. The primary reason for this is the decreasing number of alternate optimal solutions as the maximum cost increases. This gives another explanation for the maximum (minimum) cost effect.

11. Conclusions

The recursive algorithm has been described and tested, and shown to be superior for at least those problems which it is capable of solving in one pass or only slightly more than one pass. It's full comparison with other methods awaits further computational testing by the author and others. However, it is clearly a competitive algorithm for at least some kinds of assignment problems.

The author will report on further tests of the method elsewhere. Also extensions of these ideas to the solution of sum and bottleneck transportation problems will be discussed at a later time.

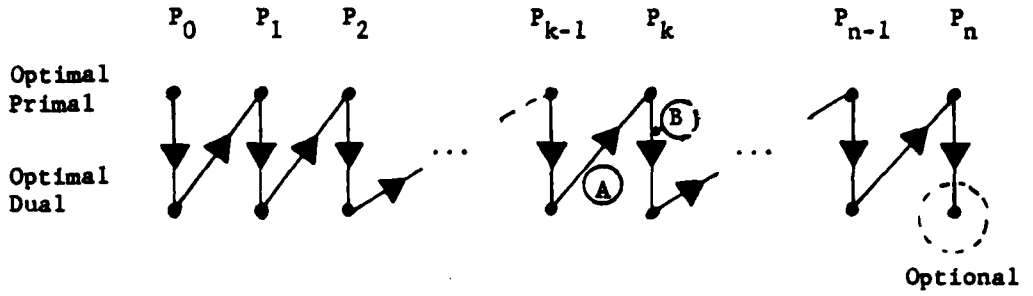


Figure 1. Order of solution of problems P_k in the recursive method.

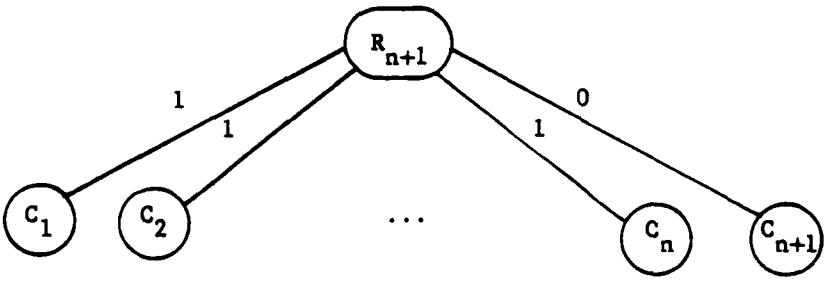


Figure 2. Optimal basis graph for P_0.

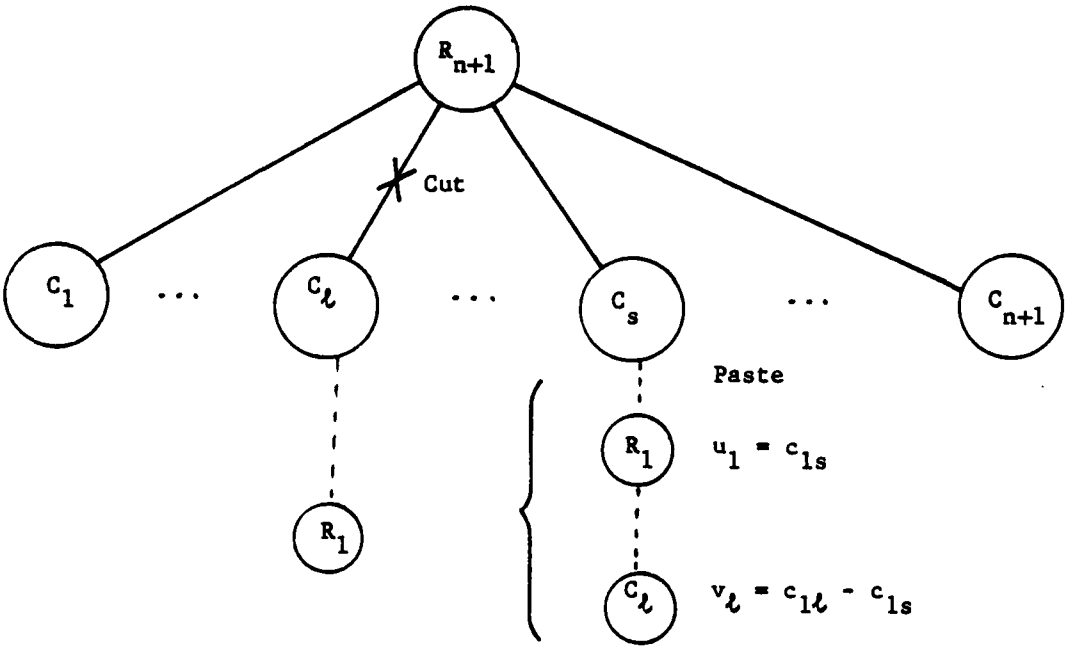


Figure 3. An Easy Back Shift. It is always possible to solve P_1 by an easy backshift.

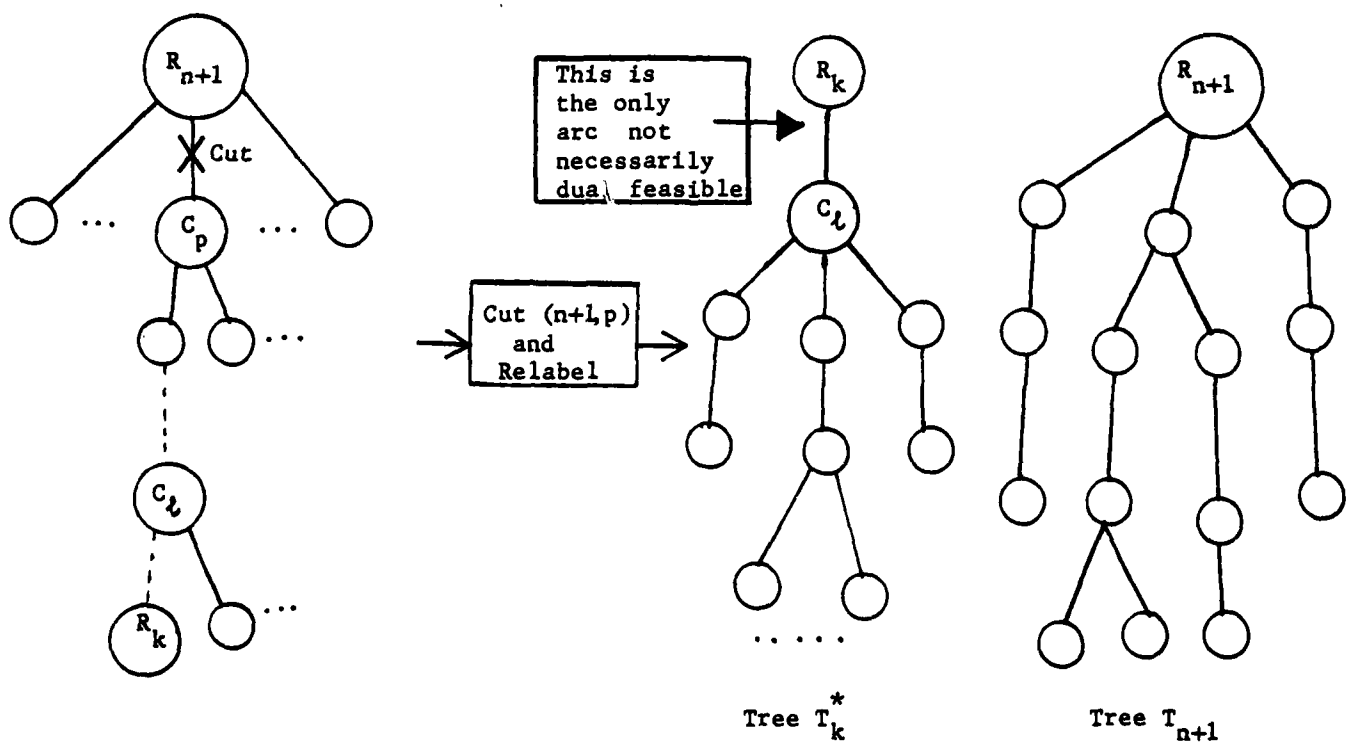


Figure 4. An Ordinary Backshift Step

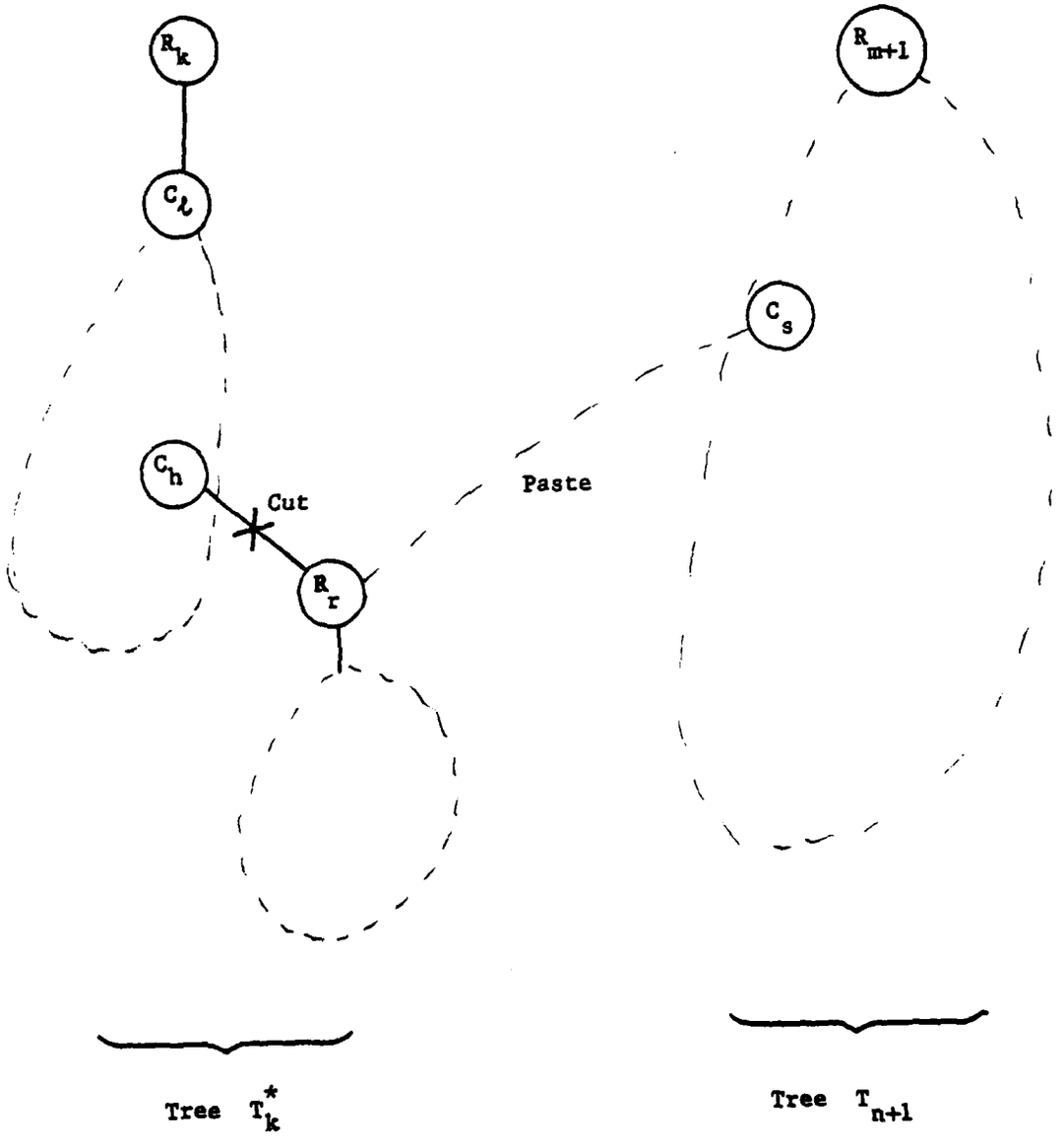
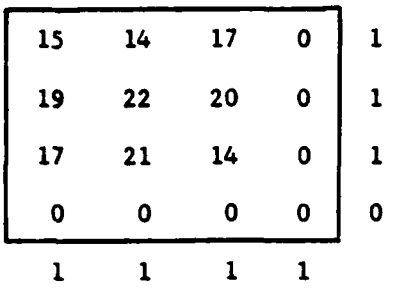
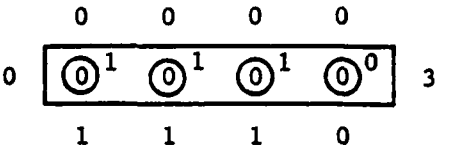


Figure 5. A Zero Shift Step.



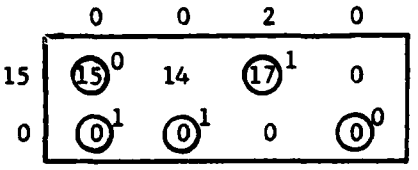
15	14	17	0	1
19	22	20	0	1
17	21	14	0	1
0	0	0	0	0
1	1	1	1	

(a) Original Problem



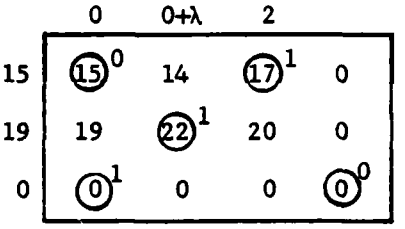
	0	0	0	0
0	(0)^1	(0)^1	(0)^1	(0)^0	3
	1	1	1	0

(b) Optimal Solution to P_0.



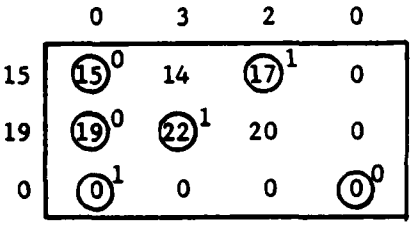
	0	0	2	0
15	(15)^0	14	(17)^1	0
0	(0)^1	(0)^1	0	(0)^0

(c) Optimal Solution to P_1.



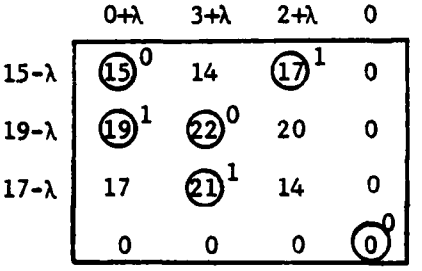
	0	0+λ	2	
15	(15)^0	14	(17)^1	0
19	19	(22)^1	20	0
0	(0)^1	0	0	(0)^0

(d) End of Backpivot Step for P_2



	0	3	2	0
15	(15)^0	14	(17)^1	0
19	(19)^0	(22)^1	20	0
0	(0)^1	0	0	(0)^0

(e) Optimal Solution to P_2.



	0+λ	3+λ	2+λ	0
15-λ	(15)^0	14	(17)^1	0
19-λ	(19)^1	(22)^0	20	0
17-λ	17	(21)^1	14	0
	0	0	0	(0)^0

(f) End of Backpivot Step for P_3. Note that the optimal assignments are now determined.

Figure 6. Solution of an Example

	$15+\lambda$	$18+\lambda$	17	0
0	15	14	$\textcircled{17}^{1}$	$\textcircled{0}^{0}$
$4-\lambda$	$\textcircled{19}^{1}$	$\textcircled{22}^{0}$	20	0
$2-\lambda$	17	$\textcircled{21}^{1}$	14	0
0	0	0	0	$\textcircled{0}^{0}$

(g) End of first zero shift

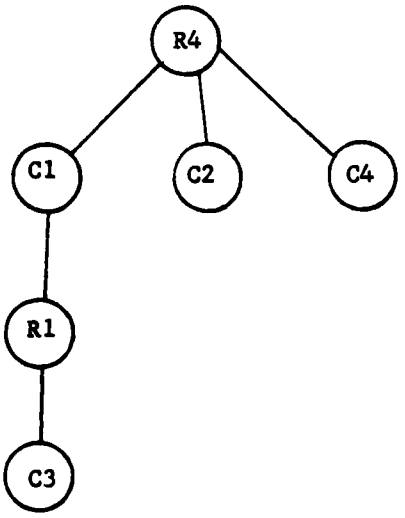
	16	$19+\lambda$	17	0
0	15	14	$\textcircled{17}^{1}$	$\textcircled{0}^{0}$
3	$\textcircled{19}^{1}$	22	$\textcircled{20}^{0}$	0
$1-\lambda$	17	$\textcircled{21}^{1}$	14	0
	0	0	0	$\textcircled{0}^{0}$

(h) End of second zero shift

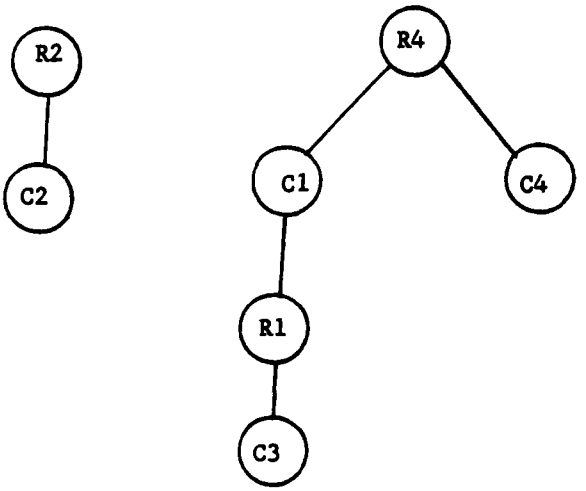
	16	20	17	0
0	15	14	$\textcircled{17}^{1}$	$\textcircled{0}^{0}$
3	$\textcircled{19}^{1}$	22	$\textcircled{20}^{0}$	0
1	$\textcircled{17}^{0}$	$\textcircled{21}^{1}$	14	0
0	0	0	0	$\textcircled{0}^{0}$

(i) End of third zero shift.
Optimal Solution to P_3.

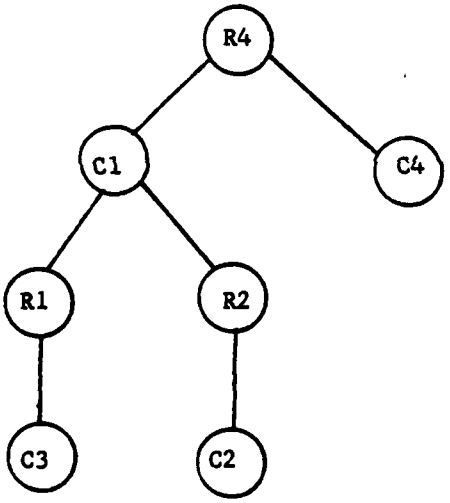
Figure 6 (continued)



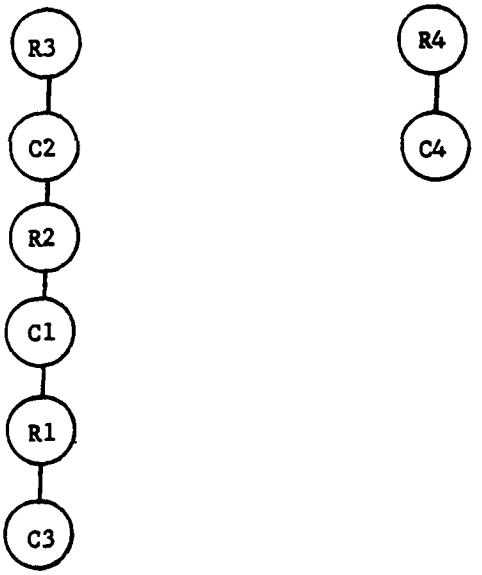
(a) Optimal basis tree for P_1



(b) End of backpivot step for P_2

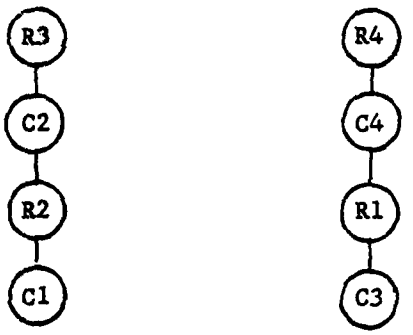


(c) Optimal basis tree for P_2

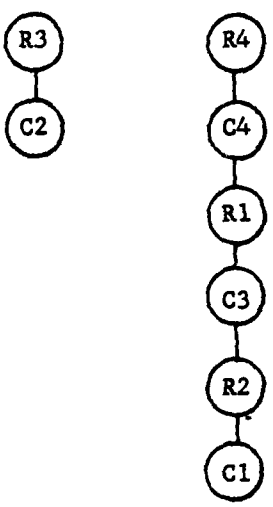


(d) End of backpivot step for P_3

Figure 7



(e) End of first zero shift



(f) End of second zero shift

(g) End of third zero shift.
Optimal basis tree for P_3.

Figure 7

	Number of Easy Backshifts	Number of Ordinary Backshifts	Number of Zero Shifts
Best Case	n-1	1	1
Worst Case	1	n-1	$\frac{(n+2)(n-1)}{2}$

Figure 8. Number of operations for the best and worst cases.

Problem Defined In Equation	Easy Backshifts	Zero Shifts	Easy Zero Shifts	Area Search Factor	Time DEC-20 Secs.
(22)	99	1	0	1.00	.26
(23)	1	5049	4949	1.51	2.99
(24)	1	99	0	34.48	8.22
(25)	1	5049	0	859.48	166.20

Figure 9. Best and Worst Case Examples

Ford-Fulkerson Dual Method [5]	n^2-1 non-break throughs
Balinski-Gomory Primal Method [1]	n^2 non-break throughs
Srinivasan-Thompson Cost Operator Method [10]	n(2n+1) primal pivots
Recursive Method	$\frac{(n+2)(n-1)}{2}$ zero shifts.

Figure 10. Worst case bounds for various polynomially bounded algorithms.

	Primal Pivot	Backshift	Zero Shift
Find incoming arc	Area search	Row search	Area search
Find cycle	Yes	No	No
Find outgoing arc	Yes	No	No
Cut and paste	Yes	Yes	Yes
Relabel	Yes	Yes	No
Change duals	Yes	No	Yes (No for an easy zero shift)

Figure 11. Comparison of steps in a primal pivot with those in a backshift or zero shift.

Maximum cost	n	Average Number of Arcs	Percentage of Easy Backshifts	Number of Zero Shifts	Percentage of Easy Zero Shifts	Area Search Factor	Time (DEC-20 Secs.)
1	100	3980	90.4	81.8	85.46	1.08	.14
	200	3968	82.5	263.6	85.5	1.25	.27
	300	4267	78.1	630.6	89.36	1.60	.54
2	100	4002	87.0	100.8	81.93	1.14	.16
	200	4017	76.3	396.2	86.94	1.56	.34
	300	4278	64.9	858.6	86.75	2.42	.75
5	100	3973	73.8	144.4	80.91	1.39	.20
	200	3981	59.0	650.0	86.75	3.83	.74
	300	4256	54.3	1304.4	88.84	8.41	1.34
10	100	3994	60.6	233.4	86.68	2.79	.42
	200	4016	48.7	658.8	82.19	7.38	.93
	300	4298	47.6	1104.4	82.84	11.57	1.88
20	100	4003	53.2	275.2	80.52	5.74	.61
	200	4016	45.7	721.0	79.00	11.66	1.43
	300	4321	44.9	1078.6	77.36	15.24	2.18
50	100	3999	45.6	281.4	69.59	10.04	.97
	200	4023	44.9	586.2	64.73	16.24	1.92
	300	4336	43.7	889.0	60.21	20.70	2.88
100	100	4025	44.0	255.4	57.15	12.38	1.12
	200	4000	43.3	595.4	49.02	24.23	2.74
	300	4263	43.2	925.6	47.98	31.45	4.30
200	100	3997	43.8	275.4	46.54	17.79	1.63
	200	4019	42.0	525.8	35.48	28.65	3.09
	300	4308	42.7	882.4	35.16	43.06	5.74
500	100	3992	46.6	234.4	23.97	19.91	1.89
	200	3997	43.4	535.8	20.29	37.7	4.05
	300	4314	43.5	864.8	20.18	52.8	6.47
1000	100	4016	42.6	234.4	16.87	21.97	1.91
	200	4003	42.5	571.2	11.42	44.74	4.82
	300	4330	43.0	842.6	11.44	58.17	7.60

Table 1. Computational experience for problems having approximately 4000 arcs. Each row presents average results for five randomly generated test problems.

n	Average Number of Arcs	Maximum Cost	Percentage of easy Backshifts	Number of Zero Shifts	Percentage of Easy Zero Shifts	Area Search Factor	Time (DEC-20 Secs.)
200	17948	1 100	96.8 44.9	106.4 679.8	74.11 77.32	1.03 12.17	.47 4.72
300	18009	1 100	94.27 46.47	230.8 1000.0	89.68 74.14	1.05 16.24	.63 6.61
400	17917	1 100	93.3 45.25	362.8 1457.0	90.56 72.44	1.07 22.43	.82 9.99
500	18063	1 100	90.8 44.28	628.0 1893.4	92.32 71.45	1.12 25.46	1.14 12.36
600	18048	1 100	88.9 43.67	1000.6 2095.0	92.68 67.37	1.18 27.78	1.49 14.73
700	18704	1 100	86.2 43.14	1304.6 2690.2	92.32 67.70	1.22 35.27	1.96 19.64
800	18627	1 100	83.68 43.50	1745.0 2952.8	92.07 66.04	1.29 36.07	2.42 20.87
900	18292	1 100	81.82 42.33	2191.8 3402.0	92.26 64.70	1.36 39.84	3.00 24.21
1000	17976	1 100	79.66 42.82	2958.4 3877.8	92.93 64.00	1.49 46.34	3.66 31.17

Table 2. Computational experience for large problems having about 18000 arcs. Each row presents average results for five randomly generated test problems.

Step	Easy Back Shifts	Ordinary Back Shifts	Zero Shifts	Easy Zero Shifts
92	92	0	0	0
93	92	1	1	0
94	92	2	2	0
95	93	2	2	0
96	93	3	3	0
97	94	3	3	0
98	95	3	3	0
99	95	4	23	19
100	95	5	25	19

Table 3. Cumulative data from the solution of a randomly generated 100 x 100 problem with 5028 arcs and maximum cost 1. The area search factor was 1.04.

Bibliography

[1] Balinski, M. L. and R. E. Gomory, "A primal method for the assignment and transportation problems," Management Science 10(1964) 578-593.

[2] Barr, R. S., F. Glover, and D. Klingman, "The Alternating Path Algorithm for Assignment Problems," Mathematical Programming, 13(1977) 1-13.

[3] Barr, J. L. and T. L. Shaftel, "Solution Properties of Deterministic Auctions," Journal of Financial and Quantitative Analysis, (1976) 287-311.

[4] Derigs, U. and V. Zimmermann, "An Augmenting Path Method for Solving Linear Bottleneck Transportation Problems," Computing, 22(1979) 1-15.

[5] Ford, L. R., and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N. J., 1962.

[6] Hung, M. S. and W. O. Rom, "Solving the Assignment Problem by Relaxation," Working Paper 78-5, 1978, College of Business Administration, Cleveland State University.

[7] Shapley, L. S. and M. Shubik, "The Assignment Game I: The Core," International Journal of Game Theory, 1(1972), 111-130.

[8] Srinivasan, V. and G. L. Thompson, "Accelerated Algorithms for Labelling and Relabelling of Trees, with Applications to Distribution Problems," Journal of the Association for Computing Machinery, 9(1972), 712-726.

[9] Srinivasan, V. and G. L. Thompson, "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," Journal of the Association for Computing Machinery, 20(1973), 194-213.

[10] Srinivasan, V. and G. L. Thompson, "Cost Operator Algorithms for the Transportation Problem," Mathematical Programming, 12(1977), 372-391.

[11] Thompson, G. L., "Computing the Core of a Market Game," to appear in the Proceedings of the Sumposium on Extremal Methods (in honor of A. Charnes).

[12] Thompson, G. L. "Pareto Optimal Deterministic Models for Bid and Offer Auctions," in Quantitative Planning and Control, Y. Ijiri and A. Whinston, Eds.. Academic Press, New York. 1979, pp. 125-140.